Sorting algorithms implemented

1.bubble sort

2.insertion sort

3.selection sort

4.merge sort

5.quick sort(last)

6.quick sort(median)

7. heap sort

Main data structure used – python list

* All sorting algorithms operate on python list, which is mutable, dynamic array.
* List allow random access and in-place updates, making them good choice for implementing sorting techniques.

Functionality-

* Each sorting algorithm is implemented as aseparate function.
* Code includes a while loop that will prompt for user input, users can enter a list of numbers or type ‘random’ to generate a random list of 500 numbers or type ‘q’ or ‘Q’ to quit.
* Error handling is included to catch invalid user input i.e. non- integer values
* Also time.perf-counter() function is used to measure the execution time of each sorting algorithm.

Algorithm details-

1.bubble sort-

Function used - bubble\_sort(num)

* Program iterates through the list, compares adjacent elements, and swap
* Outer loop iterates n times (i.e. length of the list)
* Inner loop compares adjacent elements and swaps them if they are in wrong order
* The largest element will be to the end of the list after each pass/iteration.
* Simple but inefficient for large datasets.

2.insertion sort-

Function use - insertion\_sort(arr)

* builds the sorted list one element at a time by iteratively taking next element and insertion it into its correct position in the sorted part.
* iterates through the list, picking one element as key in each pass.
* shifts elements greater than the key to the right
* inserts the key into its correct position
* efficient for small or nearly sorted dataset

3.selection sort-

Main function - selection\_sort(my\_list)

* finds the minimum element from the unsorted part and places it at the beginning.
* repeats until the entire list is sorted.
* efficiency is o(n^2) regardless of input order.

4.merge sort

Function used- merge\_sort(arr)

* a divide and conquer algorithm that splits the array into halves , recursively sorts each half and merge the sorted array
* recursively divides the list into two halves until each sublist has one element.
* merges two sorted sublists into one sorted list by comparing elements from each list.
* merge\_two\_sorted\_list (left , right) is used to merge two sorted sublists.
* merge sort(O(nlogn)) performs good when compared to previous algorithms such as bubble sort , insertion sort and selection sort but merge sort requires extra space for merging.

5.quick sort(last)

Function used quick\_sort(apple)

* here i have selected the last element as the pivot, then partition the array around the pivot, and recursively sort the partitions
* selects the last element as the pivot
* partitions the list into two sublists-ensures all elements left of pivot are less than it and all to the right are greater than the pivot.
* recursively sorts the sublists.
* efficiency is O(nlogn) but gets worse O(n^2) when the largest or smallest element is picked as pivot

6.quick sort(median)-

Functions used –

three\_median(mlist)- to find median of the first, middle ,last element.

qs\_3median(mlist) – to perform quick sort.

* what it do- improves quick sort by choosing the median of the first, middle, and last elements as pivot.
* partitions the list into sublists based on the pivot
* then recursively sorts the sublist
* advantage is - reduces the likelihood of worst-case time complexity by making pivot choice more robust.

7. heap sort

* What it does- turns the array into a max-heap , repeatedly removes the largest element , and rebuilds the heap
* swap(lst, i ,j) swaps two elements in the list
* shiftdown(lst,i, upper) maintains the heap property by shifting down an element in the heap
* heapsort(lst)- builds the max heap from the input list, repeatedly extracts the maximum element(root) and place it at the end of the sorted portion and re-heapifies the remaining elements.
* efficiency - O(nlogn), in place but not stable.

Summary-

Quick sort is typically the fastest algorithm, mainly for larger datasets followed by merge sort and heap sort.

Insertion sort, selection sort , and bubble sort are generallly slower and not recommended for large datasets

Note- quick-sort can perform worse with time complexity O(n2) if the pivot is chosen poorly.

Comparison table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Algorithms | Time complexity | stable | inplace |
| 1 | Bubble Sort | O(n2) | yes | Yes |
| 2 | Insertion sort | O(n2) | Yes | yes |
| 3 | Selection sort | O(n2) | no | Yes |
| 4 | Merge sort | O(nlogn) | yes | no |
| 5 | Quick sort(last) | O(nlogn) | no | yes |
| 6 | Quick sort(median) | O(nlogn) | No | Yes |
| 7 | Heap sort | O(nlogn) | No | yes |